CMPSC265 Data Structures and Algorithms
Homework 3

Due: Sunday, Sept 22nd, 2019, 11:59pm

Learning Objectives

- Sorting algorithms
- Implementing the Comparable and Comparator Interface
- Implementing the Stack data structure
- Applications on Stack

Deliverables:

- The modified java file *OddEvenSort.java* for Problem 1
- The modified java file *Student.java* for Problem 2
- The modified java file *StackX.java* for Problem 3
- A Java file *RemoveDuplicate.java* for Problem 4

Instructions:

- For Problem 1, 2 and 3, please first download the Java source file, modify the codes based upon requirements, and submit the updated source file.
- Please add the required *credit comments* and *comments* to each Java program you submit.

Problem Specifications:

Problem 1: OddEvenSort class (25')

Description:

Another simple sorting algorithm is the odd-even sort. The idea is to repeatedly make two passes through the array. On the first pass you look at all the pairs of items, a[j] and a[j+1], where j is odd (j = 1, 3, 5, ...). If their key values are out of order, you swap them. On the second pass you do the same for all the even values, comparing all a[j] and a[j+1] pairs (j = 2, 4, 6, ...), and do swap if they are out of order. You do these two passes repeatedly until the array is sorted. Please modify the provided OddEvenSort.java file to implement this.

Feel free to add other methods if you need.

Outputs:

Your output should look something like follows.

```
$ java OddEvenSort
The original array is:
[]
After sorting, the array is:
[]

The original array is:
[-1, 0, 100, 20, 0, 0, -2, 10, 12]
After sorting, the array is:
[-2, -1, 0, 0, 0, 10, 12, 20, 100]

The original array is:
[20, 18, 16, 14, 12, 10, 8, 6, 4, 2, 0]
After sorting, the array is:
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

The original array is:
[3, 4, 0, 5, 9, 20, 15, 14, 17, 24, 56, 0, 0, 1]
After sorting, the array is:
[0, 0, 0, 1, 3, 4, 5, 9, 14, 15, 17, 20, 24, 56]
```

Problem 2: *Student class (25')*

Description:

Please first download the attached Student.java file. You are expected to:

- Implement the *compareTo()* method to the Student class so that students can be sorted by their last name.
- Finish implementing the *FirstNameCompare* class so that students can be sorted by their first name.
- Finish implementing the *MajorCompare* class so that students can be sorted by their major.
- Finish implementing the *GPACompare* class so that students can be sorted **in descending order** of their GPA.
- Write codes in the *main()* method to sort the students by different criteria.

Outputs:

Your outputs should look something as follows.

```
$ java Student
Sort the students by their lastname:
Brown Sophie math 3.72
Davis Nele cs 3.88
Johnson Ava math 3.92
Smith Ellen cs 3.78
Williams Mia cs 3.66

Sort the students by their firstname:
Johnson Ava math 3.92
Smith Ellen cs 3.78
Williams Mia cs 3.66
Davis Nele cs 3.88
Brown Sophie math 3.72

Sort the students by their major:
Smith Ellen cs 3.78
Williams Mia cs 3.66
Davis Nele cs 3.88
Johnson Ava math 3.92
Brown Sophie math 3.72

Sort the students by their GPA (from highest to lowest):
Johnson Ava math 3.92
Davis Nele cs 3.88
Smith Ellen cs 3.78
Brown Sophie math 3.72
Williams Mia cs 3.66
```

Problem 3: StackX class (25')

Description:

Please first download the attached StackX.java file. You are expected to:

- Implement the *push()* method so that when a new data item is added to the stack that has reached its full capacity, the capacity of the original stack will be **doubled** so that the new data can be successfully added.
- Implement the *pop()* method so that when the number of data items in the stack is less than half of its capacity due to popping out items, the capacity of the stack will be shrink to half of its original capacity.

Hint: to double the capacity of the stack, you need to create a new array with doubled length, and copy all the elements in the old array to the new array. You can do similar things when shrinking half of the stack's capacity.

Outputs:

Your outputs should look something as follows.

```
$ java StackX
150 120 100 80 60 40 20
20
```

Problem 4: Remove Duplicate Characters (25')

Description

Given a string S, a duplicate removal consists of choosing two adjacent and equal letters, and removing them. We can repeatedly make duplicate removals on S until we no longer can.

Please write a program that can read in an arbitrary string entered by users, and remove all the adjacent duplicate characters in the string. The program will then output the final resulting string onto screen.

You may assume all input strings are in lower case.

Specification

Two methods are needed in this *RemoveDuplicate* class.

- main() method
- remove() method.

main() method:

1) Please read in an arbitrary string from keyboard.
2) Invoke the *remove()* method to remove all adjacent duplicate characters in the input string.
3) Output the string returned by the *remove()* method.

remove() method

- The method has one parameter, which is a string.

- The method will remove all adjacent duplicate characters in the given string.
- The method will return the resulting string.

Outputs: Your results should look similar as follows.

$ java RemoveDuplicate
```
Please enter a string: abbaca
After removing all adjacent duplicate characters, the
string is: ca
```

$java RemoveDuplicate
```
Please enter a string: abcddcb
After removing all adjacent duplicate characters, the
string is: a
```

$java RemoveDuplicate
```
Please enter a string: adddda
After removing all adjacent duplicate characters, the
string is:
```